

Boolean Functions, Quantum Gates, Hamilton Operators, Spin Systems and Computer Algebra

Yorick Hardy* and Willi-Hans Steeb†

* Department of Mathematical Sciences,
University of South Africa, Johannesburg, South Africa,
e-mail: hardyy@unisa.ac.za

† International School for Scientific Computing,
University of Johannesburg, Auckland Park 2006, South Africa,
e-mail: steebwilli@gmail.com

Abstract. We describe the construction of quantum gates (unitary operators) from boolean functions and give a number of applications. Both non-reversible and reversible boolean functions are considered. The construction of the Hamilton operator for a quantum gate is also described with the Hamilton operator expressed as spin system. Computer algebra implementations are provided.

1 Introduction

A boolean function f on n variables is a mapping $\{0, 1\}^n$ into $\{0, 1\}$. Let $x_j \in \{0, 1\}$ for $j = 1, \dots, n$. We set $\mathbf{x} = (x_1, x_2, \dots, x_n)$. In the following \cdot denotes the AND operation, $+$ denotes the OR operation, \oplus the XOR operation and \neg is the NOT operation. For $n = 1$ we have the four boolean functions $f_1(x) = 0$, $f_2(x) = 1$, $f_3(x) = x$, $f_4(x) = \bar{x}$. The last two are of course reversible.

Let $X = \{0, 1\}$ and $x_j \in X$. A boolean function \mathbf{f} with n input variables, x_1, \dots, x_n and n output variables, y_1, \dots, y_n is a function $\mathbf{f} : X^n \rightarrow X^n$ obeying

$$\mathbf{f}(x_1, \dots, x_n) \mapsto (y_1, \dots, y_n).$$

Here $(x_1, \dots, x_n) \in X^n$ is called the input vector and $(y_1, \dots, y_n) \in X^n$ is called the output vector. An n -input and n -output boolean function \mathbf{f} is reversible if it maps each input vector to a unique output vector, i.e. the map is a bijection.

Quantum gates are described by unitary operators. In the finite dimensional Hilbert space \mathbb{C}^d we have $d \times d$ unitary matrices. We describe how $2^{n+1} \times 2^{n+1}$ unitary matrices can be associated with a non-reversible boolean function f and

how $2^n \times 2^n$ unitary matrices can be associated with reversible boolean functions \mathbf{f} . Furthermore the construction of the associated Hamilton operator is derived as well as the finding of the associated spin system.

Finally computer algebra implementations in SymbolicC++ for the two cases are provided.

2 Reversible Boolean Function and Quantum Gates

Let $\mathbf{f}(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_n(x_1, \dots, x_n))$ be a reversible boolean function where $\mathbf{f} : \{0, 1\}^n \rightarrow \{0, 1\}^n$ and $f_j : \{0, 1\}^n \rightarrow \{0, 1\}$ for $j \in \{1, 2, \dots, n\}$.

We consider the standard basis in the Hilbert space \mathbb{C}^2

$$\left\{ |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}.$$

Then a corresponding $2^n \times 2^n$ permutation matrix $U_{\mathbf{f}}$ can be constructed from

$$U_{\mathbf{f}}(|x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_n\rangle) = |f_1(\mathbf{x})\rangle \otimes |f_2(\mathbf{x})\rangle \otimes \dots \otimes |f_n(\mathbf{x})\rangle$$

where $\mathbf{x} = (x_1, \dots, x_n)$. Given the permutation matrix $U_{\mathbf{f}}$ the reversible boolean function can be constructed as follows

$$\mathbf{f}(x_1, \dots, x_n) = (y_1, \dots, y_n) \quad \Leftrightarrow \quad (\langle y_1| \otimes \dots \otimes \langle y_n|) U_{\mathbf{f}}(|x_1\rangle \otimes \dots \otimes |x_n\rangle) = 1.$$

Let the function $b : \{0, 1\}^n \rightarrow \{0, 1, \dots, 2^n - 1\}$ be defined by

$$b(x_1, \dots, x_n) := \sum_{j=1}^n x_j 2^{n-j}.$$

Then

$$(U_{\mathbf{f}})_{b(x_1, \dots, x_n), b(y_1, \dots, y_n)} = \begin{cases} 1 & \mathbf{f}(x_1, \dots, x_n) = (y_1, \dots, y_n) \\ 0 & \text{otherwise} \end{cases}$$

i.e. the permutation matrix $U_{\mathbf{f}}$ has a 1 in row $b(y_1, \dots, y_n)$ and column $b(x_1, \dots, x_n)$ if and only if $\mathbf{f}(x_1, \dots, x_n) = (y_1, \dots, y_n)$, otherwise it has a 0 in that entry.

3 Examples for Reversible Boolean Gates

Example 1. Consider the reversible gate (Feynman gate)

$$x_1 \rightarrow x_1, \quad x_2 \rightarrow x_1 \oplus x_2.$$

The inverse function is given by $(x_1, x_2) \rightarrow (x_1, x_1 \oplus x_2)$. Let $|0\rangle, |1\rangle$ be the standard basis in the Hilbert space \mathbb{C}^2 . Thus we are looking for the unitary matrix which implements $(x_1, x_2 \in \{0, 1\})$

$$|x_1\rangle \otimes |x_2\rangle \mapsto |x_1\rangle \otimes |x_1 \oplus x_2\rangle.$$

We have

$$\begin{aligned} |0\rangle \otimes |0\rangle &\mapsto |0\rangle \otimes |0\rangle, & |0\rangle \otimes |1\rangle &\mapsto |0\rangle \otimes |1\rangle \\ |1\rangle \otimes |0\rangle &\mapsto |1\rangle \otimes |1\rangle, & |1\rangle \otimes |1\rangle &\mapsto |1\rangle \otimes |0\rangle. \end{aligned}$$

This provides the 4×4 permutation matrix

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \oplus \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

which is the CNOT-gate and \oplus denotes the direct sum.

Example 2. Let $x_1, x_2 \in \{0, 1\}$ and \oplus be the XOR operation. Then

$$(x_1, x_2) \mapsto (x_1 \oplus 1, x_1 \oplus x_2)$$

is a 2-bit reversible gate since

$$(0, 0) \mapsto (1, 0), \quad (0, 1) \mapsto (1, 1), \quad (1, 0) \mapsto (0, 1), \quad (1, 1) \mapsto (0, 0).$$

Let $|0\rangle, |1\rangle$ be the standard basis in \mathbb{C}^2 . To find the 4×4 permutation matrix P such that

$$P(|x_1\rangle \otimes |x_2\rangle) = |x_1 \oplus 1\rangle \otimes |x_1 \oplus x_2\rangle$$

we calculate the Kronecker products of the vectors. This provides the four equations for P

$$\begin{aligned} P \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, & P \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \\ P \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} &= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, & P \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \end{aligned}$$

Consequently we obtain the 4×4 permutation matrix

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

with the eigenvalues $+1, -1, +i, -i$.

Example 3. Given the 4×4 permutation matrix

$$U = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

with the eigenvalues $+1, -1, +i, -i$. Then we obtain the corresponding boolean function as follows. Since the matrix has $4 = 2^2$ rows, the function $\mathbf{f} : \{0, 1\}^2 \rightarrow \{0, 1\}^2$ has two arguments. The first column (i.e. the column numbered 0) has a 1 in the third row (the row numbered 2) for which $b^{-1}(0) = (0, 0)$ and $b^{-1}(2) = (1, 0)$. Thus $(0, 0) \rightarrow (1, 0)$. From the second column $(0, 1) \rightarrow (0, 0)$. The third column provides $(1, 0) \rightarrow (1, 1)$ and the fourth column $(1, 1) \rightarrow (0, 1)$. Thus we have the map

$$(0, 0) \mapsto (1, 0), \quad (0, 1) \mapsto (0, 0), \quad (1, 0) \mapsto (1, 1), \quad (1, 1) \mapsto (0, 1).$$

The right hand side provides the boolean expression

$$\mathbf{f}(x_1, x_2) = (\overline{x_1} \cdot \overline{x_2} + x_1 \cdot \overline{x_2}, x_1 \cdot \overline{x_2} + x_1 \cdot x_2) = (\overline{x_2}, x_1).$$

Example 4. Consider the reversible 3-input/3-output gate given by

$$\begin{aligned} x'_1 &= x_1 \oplus x_3 \\ x'_2 &= x_1 \oplus x_2 \\ x'_3 &= (x_1 \cdot x_2) \oplus (x_1 \cdot x_3) \oplus (x_2 \cdot x_3). \end{aligned}$$

The inverse is given by

$$\begin{aligned} x_1 &= x'_1 \cdot x'_2 \cdot \overline{x'_3} + \overline{x'_1} \cdot x'_3 + \overline{x'_2} \cdot x'_3 \\ x_2 &= \overline{x'_1} \cdot x'_2 \cdot \overline{x'_3} + x'_1 \cdot x'_3 + \overline{x'_2} \cdot x'_3 \\ x_3 &= x'_1 \cdot \overline{x'_2} \cdot \overline{x'_3} + \overline{x'_1} \cdot x'_3 + x'_2 \cdot x'_3. \end{aligned}$$

The permutation matrix takes the form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

4 Non-reversible Boolean Functions and Quantum Gates

Let $|0\rangle$, $|1\rangle$ be the standard basis in the Hilbert space \mathbb{C}^2 . Then the quantum states

$$|\mathbf{x}\rangle \equiv |x_1 x_2 \dots x_n\rangle \equiv |x_1\rangle \otimes |x_2\rangle \otimes \dots \otimes |x_n\rangle$$

form a basis in the Hilbert space \mathbb{C}^{2^n} , where $x_1, \dots, x_n \in \{0, 1\}$. We apply the ordering

$$|000 \dots 00\rangle, |000 \dots 01\rangle, |000 \dots 10\rangle, \dots, |111 \dots 11\rangle.$$

A general state $|\psi\rangle$ in the Hilbert space \mathbb{C}^{2^n} can be written as

$$|\psi\rangle = \sum_{j_1, \dots, j_n=0}^1 c_{j_1 \dots j_n} |j_1\rangle \otimes \dots \otimes |j_n\rangle.$$

It is well-known (Nielsen and Chuang [1], Hardy and Steeb [2], Steeb and Hardy [3], Gruska [4], Hirvensalo [5], Mermin [6]) that for a given boolean function f , there is a quantum circuit of comparable efficiency which computes the unitary transformation U_f which takes as input the state $|\mathbf{x}\rangle \otimes |y\rangle$ ($y \in \{0, 1\}$) in the Hilbert space $\mathbb{C}^{2^{n+1}}$ and gives the output state $|\mathbf{x}\rangle \otimes |y \oplus f(\mathbf{x})\rangle$, where \oplus is the XOR-operation. Thus we can construct a $2^{n+1} \times 2^{n+1}$ unitary matrix such that

$$U_f(|\mathbf{x}\rangle \otimes |y\rangle) = |\mathbf{x}\rangle \otimes |y \oplus f(\mathbf{x})\rangle.$$

Owing to the selection of the standard basis the unitary matrix U_f will be a permutation matrix. Vice versa given a $2^{n+1} \times 2^{n+1}$ permutation matrix, since we have a reversible boolean function, we can construct the boolean function using the same techniques as in the previous section.

If we start with the Hadamard basis in \mathbb{C}^2

$$|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad |1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

instead of the standard basis we obtain

$$U_{f,H} = (U_H \otimes \cdots \otimes U_H) U_f (U_H \otimes \cdots \otimes U_H)$$

where U_f is the permutation matrix that implements f in the standard basis and U_H is the Walsh Hadamard transform

$$U_H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = U_H^{-1}.$$

Of course, this is just a change of basis.

5 Examples for Non-Reversible Gates

Example 1. Given the boolean function $f(x_1, x_2) = x_1 \cdot \bar{x}_2$. Thus the map is $(x_1, x_2, y \in \{0, 1\})$

$$|x_1\rangle \otimes |x_2\rangle \otimes |y\rangle \mapsto |x_1\rangle \otimes |x_2\rangle \otimes |y \oplus (x_1 \cdot \bar{x}_2)\rangle$$

with

$$|0\rangle \otimes |0\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |0\rangle \otimes |0\rangle, \quad |0\rangle \otimes |0\rangle \otimes |1\rangle \mapsto |0\rangle \otimes |0\rangle \otimes |1\rangle$$

$$|0\rangle \otimes |1\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |1\rangle \otimes |0\rangle, \quad |0\rangle \otimes |1\rangle \otimes |1\rangle \mapsto |0\rangle \otimes |1\rangle \otimes |1\rangle$$

$$|1\rangle \otimes |0\rangle \otimes |0\rangle \mapsto |1\rangle \otimes |0\rangle \otimes |1\rangle, \quad |1\rangle \otimes |0\rangle \otimes |1\rangle \mapsto |1\rangle \otimes |0\rangle \otimes |0\rangle$$

$$|1\rangle \otimes |1\rangle \otimes |0\rangle \mapsto |1\rangle \otimes |1\rangle \otimes |0\rangle, \quad |1\rangle \otimes |1\rangle \otimes |1\rangle \mapsto |1\rangle \otimes |1\rangle \otimes |1\rangle.$$

This leads to the 8×8 permutation matrix

$$U_f = I_4 \oplus \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \oplus I_2$$

where \oplus denotes the direct sum and I_n is the $n \times n$ identity matrix.

Example 2. Consider the boolean function $f(x_1, x_2) = x_1 \oplus x_2$, where for the XOR operation $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. Then we have

$$|0\rangle \otimes |0\rangle \otimes |0\rangle \mapsto |0\rangle \otimes |0\rangle \otimes |0\rangle, \quad |0\rangle \otimes |0\rangle \otimes |1\rangle \mapsto |0\rangle \otimes |0\rangle \otimes |1\rangle$$

$$\begin{aligned}
|0\rangle \otimes |1\rangle \otimes |0\rangle &\mapsto |0\rangle \otimes |1\rangle \otimes |1\rangle, & |0\rangle \otimes |1\rangle \otimes |1\rangle &\mapsto |0\rangle \otimes |1\rangle \otimes |0\rangle \\
|1\rangle \otimes |0\rangle \otimes |0\rangle &\mapsto |1\rangle \otimes |0\rangle \otimes |1\rangle, & |1\rangle \otimes |0\rangle \otimes |1\rangle &\mapsto |1\rangle \otimes |0\rangle \otimes |0\rangle \\
|1\rangle \otimes |1\rangle \otimes |0\rangle &\mapsto |1\rangle \otimes |1\rangle \otimes |0\rangle, & |1\rangle \otimes |1\rangle \otimes |1\rangle &\mapsto |1\rangle \otimes |1\rangle \otimes |1\rangle.
\end{aligned}$$

This provides the 8×8 permutation matrix

$$U_f = I_2 \oplus \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \oplus \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \oplus I_2$$

where \oplus denotes the direct sum.

Example 3. Let $n = 3$. Consider the majority gate

$$f(x_1, x_2, x_3) = (x_1 \cdot x_2) + (x_1 \cdot x_3) + (x_2 \cdot x_3)$$

which returns 1 if two or three arguments of f are 1 and 0 otherwise. Then we obtain the 16×16 permutation matrix

$$U_f = I_6 \oplus \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \oplus I_2 \oplus \left(I_3 \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \right).$$

6 Construction of Hamilton Operators

For any unitary matrix V one can find a skew-hermitian matrix K with $V = e^K$. The skew-hermitian matrix can then be identified with a Hamilton operator \hat{H} (hermitian matrix times $\hbar\omega$) via $K = -i\hat{H}t/\hbar$ (Steeb and Hardy [7]). The eigenvalues of a unitary matrix are of the form $e^{i\phi}$ ($\phi \in \mathbb{R}$). Let P be an $n \times n$ permutation matrix then P^T is a permutation matrix and $PP^T = I$, i.e. $P^T = P^{-1}$. Any permutation matrix has an eigenvalue +1 with the corresponding normalized eigenvector $\frac{1}{\sqrt{n}}(1 \ 1 \ \dots \ 1)^T$. The construction of the skew-hermitian matrix K may be done via the spectral decomposition of U , i.e. we find the eigenvalues and the normalized (pairwise orthonormal) eigenvectors of U . Notice that $V = e^K = e^{K+2\pi kiI}$ for all $k \in \mathbb{Z}$, so K is not unique. More generally, if J is any matrix which commutes with K and $e^J = I$, then $e^{K+J} = V$.

Example 1. For the CNOT gate given above we obtain a skew-hermitian matrix

$$K = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \oplus \left(\frac{\pi i}{2} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \right)$$

where we utilized the spectral decomposition of U to find K .

Example 2. For the 4×4 permutation matrix

$$P = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

given above with the eigenvalues $+1, -1, +i, -i$ we obtain a skew-hermitian matrix K with $P = e^K$

$$K = \frac{\pi}{4} \begin{pmatrix} i & i & -1-i & 1-i \\ i & i & 1-i & -1-i \\ 1-i & -1-i & i & i \\ -1-i & 1-i & i & i \end{pmatrix}$$

or

$$K = -i\frac{\pi}{4} \begin{pmatrix} -1 & -1 & 1-i & 1+i \\ -1 & -1 & 1+i & 1-i \\ 1+i & 1-i & -1 & -1 \\ 1-i & 1+i & -1 & -1 \end{pmatrix}$$

with $\omega t = \pi/4$.

The Pauli spin matrices

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

together with the 2×2 identity matrix $\sigma_0 = I_2$ form an orthogonal basis in the vector space of 2×2 matrices. The spin matrices S_1, S_2, S_3 are given by $S_1 = \frac{1}{2}\sigma_1$, $S_2 = \frac{1}{2}\sigma_2$, $S_3 = \frac{1}{2}\sigma_3$. Thus each Hamilton operator in the Hilbert space \mathbb{C}^{2^n} can be written as linear combinations of Kronecker products of Pauli spin matrices $\sigma_0, \sigma_1, \sigma_2, \sigma_3$.

For example 2 given above we find the Hamilton operator

$$\hat{H} = \hbar\omega(-\sigma_0 \otimes \sigma_0 - \sigma_0 \otimes \sigma_1 + \sigma_1 \otimes \sigma_0 + \sigma_2 \otimes \sigma_0 + \sigma_1 \otimes \sigma_1 - \sigma_2 \otimes \sigma_1).$$

7 Computer Algebra Implementations

Our computer algebra implementation uses the computer algebra system SymbolicC++ [8]. In addition to the methods described above we also compute a symbolic expression for $\mathbf{f}(x_1, \dots, x_n)$, the sum of products form obtained from the

constructed truth tables (which we simplified using an implementation of resolution (Lloyd [9])).

The program illustrates Example 1 of section 5. The function `main` first finds the permutation matrix implementing the example: $f(x_1, x_2) = x_1 \cdot \overline{x_2}$. Then the map (truth table) is printed. Finally we recreate the map from the permutation matrix, which is the reversible map

$$g(x_1, x_2, x_3) = (x_1, x_2, (x_1 \cdot \overline{x_2}) \oplus x_3) = (x_1, x_2, x_1 \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot x_3 + x_2 \cdot x_3).$$

The program counts from 0, i.e. x_0 . The output is

```
[1 0 0 0 0 0 0 0]
[0 0 0 0 0 1 0 0]
[0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 1 0 0 0 0 0 0]
[0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 1]
000 -> 000
001 -> 001
010 -> 010
011 -> 011
100 -> 101
101 -> 100
110 -> 110
111 -> 111
[                x0                ]
[                x1                ]
[x0*NOT[x1]*NOT[x2]+NOT[x0]*x2+x1*x2]
```

The full program listing follows.

```
#include <bitset>
#include <iostream>
#include <list>
#include <map>
#include <vector>
#include "symbolicc++.h"
using namespace std;

const int n=3;

// a class to provide ordering of bitsets
// so that they can be used in maps
template <const size_t n>
class cmpbst
```

```

{
public:
    bool operator()(const bitset<n> &b1,const bitset<n> &b2)
    {
        size_t i;
        for(i=0;i<n;++i) if(b1[i] != b2[i]) return (b1[i] < b2[i]);
        return false;
    }
};

// for a given reversible boolean map, find the corresponding
// permutation matrix
template <const size_t n>
Symbolic permutation(const map<bitset<n>,bitset<n>,cmpbst<n> > &m)
{
    unsigned int N = (1 << n);
    Symbolic P = Symbolic("P",N,N)*0;
    typename map<bitset<n>, bitset<n> >::const_iterator i;
    for(i=m.begin();i!=m.end();++i)
        P(i->second.to_ulong(),i->first.to_ulong()) = 1;
    return P;
}

// simplifies a sum of products form using resolution the
// products are represented by bitsets and the sum is the
// list of bitsets
template <const size_t n>
list<pair<bitset<n>,bitset<n> > > simplify(const list<bitset<n> > &s)
{
    bool change = true;
    // a list which indicates whether bitsets were used in resolution
    // or need to be copied to the next round
    list<bool> copy;
    list<bool>::iterator ci1, ci2;
    // each bitset is stored with a mask which indicates which bits
    // may be used for resolution, once a bit is used it will be masked
    list<pair<bitset<n>,bitset<n> > > r, t1, t2, *tp1 = &t1, *tp2 = &t2, *tpp;
    typename list<bitset<n> >::const_iterator li;
    typename list<pair<bitset<n>, bitset<n> > >::const_iterator lpi1, lpi2;

    for(li=s.begin();li!=s.end();++li)
    {
        t1.push_back(make_pair(*li,bitset<n>()));
        // initially all bitsets propagate

```

```

    copy.push_back(true);
}
while(!tp1->empty())
{
    // track whether resolution has been applied
    // if no change is recorded, we are done
    change = false;
    for(lpi1=tp1->begin(),ci1=copy.begin();lpi1!=tp1->end();++lpi1,++ci1)
    {
        // search for a second bitset which differs from this bitset
        // in exactly one place (taking into account the masks)
        for(lpi2=lpi1,ci2=ci1;lpi2!=tp1->end();++lpi2,++ci2)
        {
            // only compare if the masks are the same
            if(lpi1->second==lpi2->second)
            {
                // XOR finds the differing bits which are then masked
                bitset<n> diff = ((lpi1->first ^ lpi2->first) & ~lpi1->second);
                // only one bit differs so apply resolution
                if(diff.count()==1)
                {
                    // mask the bit which has been used
                    tp2->push_back(make_pair(lpi1->first,lpi1->second | diff));
                    change = true;
                    // these bitsets have been used in resolution, don't copy them
                    *ci1 = *ci2 = false;
                }
            }
        }
        if(*ci1) r.push_back(*lpi1);
    }
    // reset the variables for the next application of resolution
    tpp = tp1; tp1 = tp2; tp2 = tpp; tp2->clear();
    copy.clear(); copy.resize(tp1->size(),true);
}
r.unique();
return r;
}

// find a symbolic expression for a given boolean map
template <const size_t n>
Symbolic expression(const map<bitset<n>, bitset<n>, cmpbst<n> > &m)
{

```

```

size_t j, k;
Symbolic S("S",n), NOT("NOT"), x("x",n);
vector<list<bitset<n> > > terms(n);
vector<list<pair<bitset<n>, bitset<n> > > > simplified(n);
typename map<bitset<n>,bitset<n> >::const_iterator i;
typename list<pair<bitset<n>,bitset<n> > >::iterator li;
// for each y_j, record all values of x_1,...,x_n such that y_j = 1
for(i=m.begin();i!=m.end();++i)
    for(j=0;j<n;++j) if(i->second[j]) terms[j].push_back(i->first);
// construct each symbolic expression for y_j
for(j=0;j<n;++j)
{
    S(j) = 0;
    // find a smaller set of terms
    simplified[j] = simplify(terms[j]);
    for(li=simplified[j].begin();li!=simplified[j].end();++li)
    {
        Symbolic P = 1;
        for(k=0;k<n;++k)
            if(!li->second[k])
            {
                // this is the usual construction of a product for the
                // sum of products form generated from a truth table
                if(li->first[k]) P *= x(k); else P *= NOT[x(k)];
            }
        S(j) += P;
    }
}
return S;
}

// determine the reversible boolean map from a permutation matrix
template <const size_t n>
map<bitset<n>,bitset<n>,cmpbst<n> > booleanmap(const Symbolic &permutation)
{
    size_t i, j;
    map<bitset<n>, bitset<n>, cmpbst<n> > m;
    for(i=0;i<(1<<n);++i)
        for(j=0;j<(1<<n);++j)
            if(permutation(i,j)!=0) m[bitset<n>(j)] = bitset<n>(i);
    return m;
}

// reverse the contents of a bitset

```

```

template <const int n>
bitset<n> reverse(const bitset<n> &b)
{
    size_t i;
    bitset<n> r;
    for(i=0;i<n;++i) r[n-i-1] = b[i];
    return r;
}

int main(void)
{
    int i1, i2, i3;
    bitset<3> a, b;
    map<bitset<3>, bitset<3>, cmpbst<3> > f, g;
    map<bitset<3>, bitset<3>, cmpbst<3> >::const_iterator i;
    Symbolic P;
    for(i1=0;i1<2;++i1)
        for(i2=0;i2<2;++i2)
            for(i3=0;i3<2;++i3)
            {
                a[0] = b[0] = i1; a[1] = b[1] = i2; a[2] = i3;
                b[2] = a[2]^(a[0] & (!a[1]));
                f[a] = b;
            }
    P = permutation(f);
    cout << P << endl;
    g = booleanmap<3>(P);
    for(i=g.begin();i!=g.end();++i)
        cout << reverse<3>(i->first) << " -> " << reverse<3>(i->second) << endl;
    cout << expression(f) << endl;
    return 0;
}

```

8 Conclusion

We have described the algorithms for finding the permutation matrices which implement boolean functions as a quantum gate and vice versa. The construction of the Hamilton operator and the corresponding spin system from the quantum gate is also described. Computer algebra implementations were demonstrated.

Acknowledgment

The authors are supported by the National Research Foundation (NRF), South

Africa. This work is based upon research supported by the National Research Foundation. Any opinion, findings and conclusions or recommendations expressed in this material are those of the author(s) and therefore the NRF do not accept any liability in regard thereto.

References

- [1] M. A. Nielsen and I. L. Chuang, *Quantum Computing and Quantum Information*, Cambridge University Press, Cambridge (2000)
- [2] Y. Hardy and W.-H. Steeb, *Classical and Quantum Computing with C++ and Java Simulations*, Birkhauser Verlag, Basel (2002)
- [3] W.-H. Steeb and Y. Hardy, *Problems and Solutions in Quantum Computing and Quantum Information*, World Scientific, Singapore (2012)
- [4] J. Gruska, *Quantum Computing*, McGraw-Hill (1999)
- [5] M. Hirvensalo, *Quantum Computing*, second edition, Springer, New York (2004)
- [6] N. D. Mermin, *Quantum Computer Science*, Cambridge University Press, Cambridge (2007)
- [7] W.-H. Steeb and Y. Hardy, “Quantum Gates and Hamilton Operators”, Int. J. Theor. Phys. **45**, 953-961 (2006)
- [8] Y. Hardy, Kiat Shi Tan and W.-H. Steeb, *Computer Algebra with Symbolic C++*, World Scientific, Singapore (2008)
- [9] J. Lloyd, *Foundations of Logic Programming*, second extended edition, Springer, New York (1987)